Escuela
Politécnica
Superior

Estimación de la pose de la mano en 3D a partir de imágenes 2D

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Alejandro Bañuls Ordóñez
Tutor/es:
Miguel Cazorla Quevedo
Francisco Gómez Donoso

Mayo 2020

Universitat d'Alacant
Universidad de Alicante

Estimación de la pose de la mano en 3D a partir de imágenes 2D

Autor
Alejandro Bañuls Ordóñez

Tutor/es
Miguel Cazorla Quevedo
Departamento de Ciencia de la Computación e Inteligencia Artifical
Francisco Gómez Donoso
Departamento de Ciencia de la Computación e Inteligencia Artifical

Grado en Ingeniería Informática

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Mayo 2020

Dedicado a mis padres por siempre confiar en mi,
a mi hermano por ser una fuente de inspiración y al resto de mi familia.

A mis tutores Miguel y Fran por ayudarme tanto en este camino.

Y por supuesto a mis amigos que tantas veces me han hecho relajarme del trabajo.

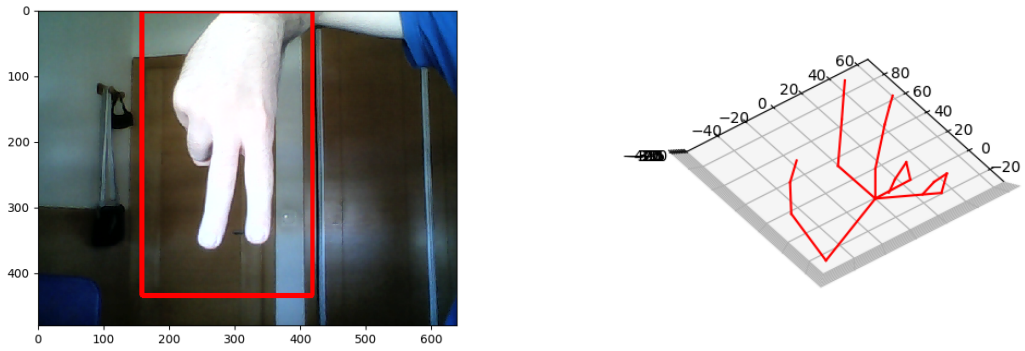
Índice general

Índice de figuras

Índice de tablas

1 Introducción

El problema que estamos tratando de resolver en este trabajo consiste en predecir la pose de una mano en 3D a partir de una imagen 2D capturada con cualquier tipo de cámara mediante Deep Learning. Con lo cual no solo se podría utilizar con imágenes sino con vídeo tanto en vivo como no, ya que simplemente habría que ir sacando frame a frame y prediciendo la pose de dicho frame. Esta predicción que hacemos consiste en identificar las coordenadas x, y, z de 21 puntos que conforman la mano, por lo que también podemos decir que el problema es un problema de regresión ya que aquí no hay clases ni categorías, (aunque más adelante se explicará unos experimentos utilizando clusters), sino que aquí la red tiene que generar 63 valores. Como vemos en 1.1 la webcam captura la imagen, detecta la mano en dicha imagen y predice los valores de la pose, que para apreciarlo mejor, los representamos en un gráfico 3D.



(a) Mano capturada por la webcam

(b) Predicción de la pose en 3D

Figura 1.1: Ejemplo de funcionamiento

Este problema es importante ya que como explicaremos más adelante hoy en día tiene muchas posibles aplicaciones y los sistemas que hacen cosas parecidas tienen desventajas que este modelo puede eliminar. Además que a día de hoy los sistemas parecidos no son accesibles por todo el mundo por lo que es necesario una mejora de estos sistemas tanto a nivel de funcionamiento (más precisión, menos tiempo de respuesta, etc) como a nivel de hardware (menos sensores,etc).

La ventaja que tiene este modelo es que se podría realizar con cualquier tipo de cámara ya que lo único que se necesita, como hemos dicho antes, es capturar los frames, pasárselos a la red neuronal y que prediga la pose, otra ventaja que tiene es que no requiere sensores específicos ni elementos externos para funcionar lo que hace que pueda ser utilizado de manera sencilla y por cualquier persona con una cámara y un ordenador.

Las aplicaciones de este modelo podrían ir desde el apartado de videojuegos, usando este para juegos, tanto de realidad virtual, con la ventaja de no necesitar ningún elemento intrusivo como guantes, como en juegos que no sean de realidad virtual, pero que con el uso de una cámara puedas interactuar con lo que va apareciendo en pantalla con precisión. También se podría utilizar en temas de rehabilitación donde al realizar los ejercicios te podría decir si estas haciendo los ejercicios mal o bien sin necesidad de tener a un fisioterapeuta contigo al lado, o cuanto has mejorado, cuanto movimiento has ganado de una sesión a otra, etc. Otra de las posibles aplicaciones que tendría este modelo sería su uso en realidad virtual pero no en juegos, sino en otros ámbitos como podría ser la medicina, donde los médicos podrían entrenar las operaciones que van a hacer. Estos necesitan un sistema preciso además de que con este modelo usarían sus propias manos que es lo que usarían en el mundo real.

Este trabajo se ha inspirado en (Gomez-Donoso y cols., 2018) y en (Gomez-Donoso y cols., 2019a).

2 Marco Teórico

2.1 Estimación de la pose

Varios trabajos han abordado el problema de la estimación de la pose de la mano en 3D y aunque podemos encontrar los primeros trabajos en la década de los noventa sigue siendo hoy en día un problema para la comunidad de la visión por computador. Los primeros trabajos en estimación de la pose de la mano se basan en un *pipeline* de visión por computador tradicional utilizando técnicas de *machine learning* tradicional. En el trabajo de (Stenger y cols., 2004) usaron clasificadores dispuestos en una estructura jerárquica para reconocer diferentes clases de objetos. Un descriptor se extrae para entrenar a los clasificadores. Estos enfoques tradicionales tienen dificultades en estimar la poses precisas. Además estos métodos no generalizan bien en el mundo real y computacionalmente hablando son caros.

Con el bajo coste de las cámaras de profundidad, los nuevos enfoques empezaron a utilizar este tipo de sensores 3D. Se ha abordado este problema usando la información de imágenes de profundidad y RGB. Con técnicas basadas en la profundidad encontramos dos enfoques:

- Basado en seguimiento de modelos 3D (de La Gorce y cols., 2011). Un modelo sintético de mano 3D se usa normalmente para generar hipótesis que se evalúan contra datos 3D reales. Las poses tienen que ser muy precisas además de que tiene el problema de tener mucho ruido, y el mayor problema son los cambios entre frames, cosa que es habitual en el mundo real y algo a lo que es muy sensible este modelo.

- Basado en técnicas discriminatorias utilizando información de profundidad. Los trabajos basados en este enfoque predicen principalmente la pose de la mano directamente desde imágenes RGB-D. Por ejemplo, (Kuznetsova y cols., 2013) utiliza *Random Forests* para la predicción. (Tang y cols., 2013) utiliza *Random Forests* para la clasificación de la máscara de la mano utilizando como entrada imágenes de profundidad. De esta manera, son capaces de detectar partes de la mano, y finalmente, en base a esta estimación, el sistema predice las ubicaciones de los *joints* 2D. Este enfoque es computacionalmente caro debido al uso de técnicas de clasificación por píxel y también requiere un gran conjunto de datos de entrenamiento para resolver la discretización.

En lo últimos años, con el aumento del uso de las *Convolutional Neural Network (CNN)* y de las técnicas de *deep learning*, han aparecido nuevos enfoques a este problema. La mayoría de estas nuevas técnicas aprovechan las CNN para la extracción automática de características de las imágenes de profundidad (Tompson y cols., 2014). Otros trabajos combinan la extracción automática de características con entrenamiento supervisado para predecir la pose de la mano 3D (Oberweger y cols., 2015). Estos sistemas necesitan una gran cantidad de datos etiquetados para entrenar donde las etiquetas corresponden con los *joints* de la mano en 3D.

El enfoque presentado en (Sinha y cols., 2016) está basado en la utilización de arquitecturas de *deep learning* entrenadas con datos sintéticos para extraer características profundas. Estas características se organizan de acuerdo con su vecindad espacial y temporal, logrando una estimación de pose de mano 3D robusta y relativamente libre de fluctuaciones. Este enfoque también funciona en mapas de profundidad y espera que el área de la imagen donde se encuentra la mano se detecte previamente. Otro enfoque de aprendizaje profundo se describe en (Tompson y cols., 2014). Su sistema utiliza una CNN para calcular una pose de la mano en 3D que luego se redefine utilizando una pose de la mano sintética predefinida. Aunque la mayoría de estos métodos comienzan a funcionar bien para la estimación de la pose de la mano en el aire, todos requieren un uso de la información de profundidad obtenida usando un sensor 3D ubicado a poca distancia.

(Zimmermann y Brox, 2017b) aborda este problema usando una sola imagen RGB y CNN. Se utilizan tres arquitecturas CNN diferentes en este enfoque. La primera realiza la segmentación de manos del usuario a nivel de píxel, la segundo detecta *joints* 2D en el espacio de la imagen y, finalmente, la tercera está entrenada para levantar los puntos clave detectados a un espacio de coordenadas 3D. Este enfoque está entrenado usando datos sintéticos y por lo tanto no proporciona resultados precisos para datos reales. Además, el uso de tres CNN independientes, incluida una red para la segmentación de manos, limitan su rendimiento. Recientemente, se han publicado otros trabajos que abordan el problema de estimación de pose manual al usar cámaras RGB de múltiples vistas ((Panteleris y Argyros, 2017);(Simon y cols., 2017)). Sin embargo, estos métodos son sensibles a los procedimientos de calibración de la cámara y requieren una estimación de profundidad previa o, al menos, triangulación 3D a través de múltiples cámaras RGB (Simon y cols., 2017).

Otros trabajos que nos encontramos utilizando imágenes RGB son por un lado, (Bilbeisi, 2019) el cual utiliza imágenes de profundidad y deep learning para sacar la pose de la mano, el principal inconveniente de este modelo es que al tratarse de imágenes de profundidad se necesita un sensor específico que detecta esta profundidad lo que desde el punto de vista económico es un incoveniente además de, al tratar con datos 3D, ser más costoso computacionalmente hablando.Y por otro lado tenemos (Ge y cols., 2019), el cual no solo se centra en estimar los *joints* 3D de la mano sino que proponen un método basado en *Graph Convolutional Neural Network* (Graph CNN) para reconstruir una malla 3D completa de la forma de la mano.

Como hemos visto, la mayoría de estos modelos requieren de un costo muy elevado computacionalmente hablando además de que ninguno de ellos ha demostrado tener una precisión muy elevada. Otro problema que nos encontramos es que muchos de ellos utilizan datos de entrenamiento sintéticos y a la hora de utilizarlos en el mundo real no funcionan, y los que no usan datos sintéticos, no generalizan bien, por lo que les pasa lo mismo, al usarlos en un entorno diferente al de los datos de entrenamiento no funciona.

2.2 Dataset

Como hemos dicho anteriormente este problema es uno de los más comunes dentro del área de la visión por computador. Primero, se aplicaron algoritmos de visión por computadora tradicional en imágenes de color estático para segmentar la mano del fondo. Luego, se utilizaron métodos de aprendizaje automático para abordar el problema y, por último, se han usado técnicas de *deep learning* para abordar el problema. Junto con estos métodos, aparecieron diferentes datasets, destinados a alimentar estos sistemas con los datos adecuados. En esta sección, revisamos estos datasets.

A principios de los 2000 aparecieron dos datasets, uno para detección de poses estáticas (Marcel y Bernier, 1999) y otro para clasificación de gestos dinámicos (Marcel y cols., 2000):

- **Dataset pose estática:** consiste en 3000 imágenes de manos con poses estáticas divididas en 6 clases diferentes.
- **Dataset gestos dinámicos:** está compuesto de 60 secuencias de manos haciendo ciertos gestos y están divididos en 4 clases.

En ambas versiones el etiquetado es la clase de gesto sin ninguna anotación de la pose y los ejemplos están muy desbalanceados lo que hará que baje la precisión de los algoritmos de entrenamiento. Además la resolución y la cantidad de ejemplos es insuficiente.

Más tarde, en 2014, apareció otro dataset de clasificación de pose de la mano (Pisharady y cols., 2014). El dataset de poses de la mano *NUS* consiste en 10 clases de poses con 24 imágenes por cada clase las cuales son capturadas variando la posición y el tamaño de la mano, pero el trasfondo de las imágenes de este dataset es uniforme. Nuevamente, este dataset no proporciona anotaciones de los *joints* de la mano, pero si una clase para cada muestra, por lo que no está destinado a ser utilizado en estimación continua de la pose de la mano. Por otra parte, el fondo uniforme se convierte en un problema para los algoritmos de aprendizaje.

El dataset introducido en (Grzejszczak y cols., 2016) contiene gestos de lenguaje de signos polaco y estadounidense. El dataset consiste en tres series de gestos que incluyen los siguientes datos: imágenes RGB con diferentes resoluciones pero con una muy buena calidad, el *ground truth* de la máscara binaria cuando haya piel y la localización de los puntos importantes de la mano, en total contienen más de 1600 ejemplos. A pesar de la alta resolución y la gran calidad del *ground truth*, este dataset no tiene suficientes ejemplos. Hay un inconveniente mayor que es que el fondo de las imágenes es siempre el mismo lo que podría reducir la capacidad de generalización del sistema ya que en un escenario real el fondo está cambiando constantemente y como ya hemos mencionado es un problema para los algoritmos de aprendizaje.

Otro dataset fue publicado por (Molina y cols., 2013). Está compuesto por mapas de profundidad naturales y sintéticos. Cada mapa de profundidad representa una mano segmentada en una pose determinada. Las anotaciones incluyen la posición 2D de cada *joint* en el frame de coordenadas del mapa de profundidad, y el valor de profundidad en ese punto. Incluyeron 65 muestras naturales dividido en 6 diccionarios diferentes divididos por su nivel de oclusión, pose o movimiento. Los datos sintéticos fueron generados tomando una semilla y realizando

hasta 200 rotaciones al azar. Este es un dataset de pose de mano altamente representativo, pero tiene un importante inconveniente: solo proporciona mapas de profundidad, lo que obliga a quien quiera usarlo a adquirir un sensor 3D. El número de muestras también es insuficiente para los algoritmos de *deep learning* y las muestras sintéticas no representan el mundo real.

Siguiendo la tendencia de profundidad estimulada por la aparición de sensores de alcance de bajo coste, se creó el dataset NYU Hand Pose Tompson y cols. (2014). Este dataset contiene 8252 conjuntos de test y 72757 frames de entrenamiento de datos RGBD además de la información del *ground truth* de la pose de la mano. Para cada frame, se proporcionan los datos RGBD de 3 Kinects: una vista frontal y 2 vistas laterales. El conjunto de entrenamiento contiene muestras de un solo usuario, mientras que el conjunto de prueba contiene muestras de dos usuarios. Una recreación sintética de la pose de la mano también se proporciona para cada vista. El principal inconveniente de este dataset es que las imágenes RGB proporcionadas son proyecciones de las nubes de puntos obtenidas por un sensor 3D. Esto hace que las imágenes en color 2D se procesen en baja calidad, con un muchas regiones desconocidas fuera del alcance del sensor 3D. Por lo tanto, para usar este conjunto de datos también requiere un sensor 3D.

Se propone un dataset diferente en (Zhang y cols., 2016). En este trabajo, los autores proponen un dataset estéreo que proporciona 18000 pares de poses estéreo de manos y su *ground truth* de las articulaciones 3D y mapas de profundidad. A pesar de la calidad de las anotaciones, este dataset carece de variabilidad. Sólo incluye una mano en la escena que se limita a contar con el movimiento de la mano, o movimientos similares al azar. Además, solo presenta dos individuos diferentes.

Como hemos visto, estos datasets tenían el problema que muchos usaban manos sintéticas lo que provocaba un mal funcionamiento con datos reales, otros tenían el mismo fondo o las muestras eran muy pequeñas lo que hacía que generalizar fuese imposible y funcionase mal con datos fuera del conjunto de entrenamiento. Además muchos de estos datasets no tenían anotaciones de los *joints* por lo que era imposible estimar su pose.

3 Objetivos y motivación

La motivación personal para el desarrollo de este proyecto se basa en:

- Aprendizaje de Deep Learning.
- Aprendizaje aplicado sobre CNNs configuradas en modo regresión.
- Aprendizaje del uso de Keras y Tensorflow

Los objetivos establecidos en este proyecto son los siguientes:

- Desarrollo de una red que estime la pose final de la mano.
- Funcionamiento de la estimación en tiempo real.
- Selección de la mejor arquitectura para resolver el problema.

4 Metodología

En este proyecto se han utilizado las siguientes herramientas:

4.1 Tensorflow

TensorFlow es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, y desarrollado por Google para satisfacer las necesidades de sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos.

4.2 Keras

Keras es una API de redes neuronales de alto nivel escrita en python que puede funcionar sobre Tensorflow. Está diseñada para permitir una rápida experimentación con redes neuronales y entre sus principales virtudes se encuentra la modularidad, la extensibilidad y el ser fácil de usar para los usuarios.

En este proyecto se ha utilizado Keras para diseñar la red neuronal, ya que como se explicará más adelante se han utilizado las arquitecturas de redes predefinidas en Keras y se han entrenado dando diferentes resultados dependiendo de cuáles se usaban. Las arquitecturas utilizadas han sido las siguientes:

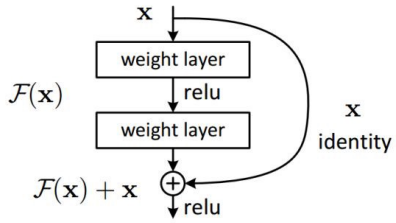


Figura 4.1: Cómo funciona ResNet.

- **ResNet50**: es un tipo de arquitectura que resuelve el problema del desvanecimiento del gradiente. Esto se debe a que cuando la red es demasiado profunda, los gradientes desde donde se calcula la función de pérdida se reducen a cero después de varias aplicaciones de la regla de la cadena. Esto provoca que la red nunca actualiza los pesos a un determinado nivel y, por lo tanto, no se realiza ningún aprendizaje. Con ResNet, los gradientes pueden pasar directamente a través de las conexiones de salto hacia atrás desde las

capas posteriores hasta los filtros iniciales como vemos en 4.1. ResNet50 en concreto tiene 50 capas.

- **Inception**: este modelo está formado por bloques de construcción simétricos y asimétricos que incluyen convoluciones, reducción promedio, reducción máxima, concatenaciones, *pooling* y capas completamente conectadas. La normalización por lotes se usa con frecuencia en todo el modelo y se aplica a las entradas de activación. Las pérdidas se calculan a través de Softmax.

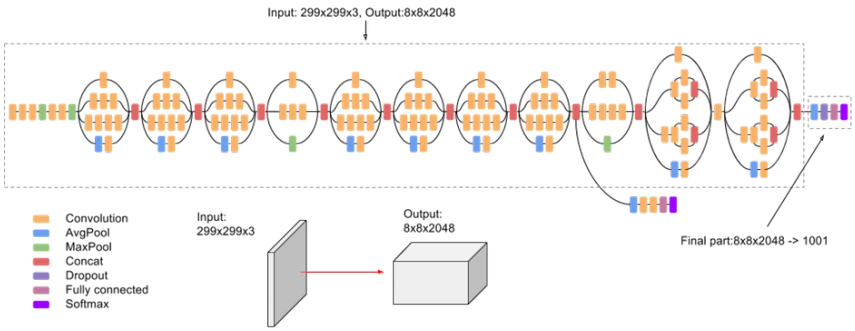


Figura 4.2: Arquitectura de la Inception

- **DenseNet**: esta arquitectura requiere menos parámetros que una CNN tradicional equivalente, ya que no es necesario aprender mapas de funciones redundantes. Además, algunas variaciones de ResNets han demostrado que muchas capas apenas contribuyen y pueden descartarse. De hecho, el número de parámetros de ResNets es grande porque cada capa tiene sus pesos para aprender. En cambio, las capas DenseNets son muy "estrechas" y solo agregan un pequeño conjunto de nuevos mapas de características. Otro problema con las redes muy profundas fueron los problemas para entrenar, debido al flujo mencionado de información y gradientes. DenseNets resuelve este problema ya que cada capa tiene acceso directo a los gradientes desde la función de pérdida y la imagen de entrada original.

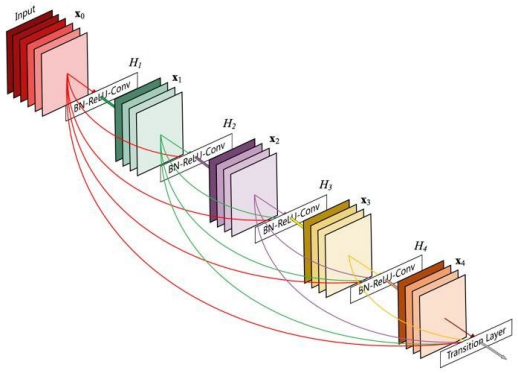


Figura 4.3: Estructura de una DenseNet

4.3 YOLO

YOLO es una red de región que permite la detección de objetos en tiempo real. Es rápida y precisa y puede llegar a procesar imágenes a 30 FPS. Además puedes compensar entre velocidad y precisión simplemente cambiando el tamaño del modelo sin necesidad de reentrenarlo.

Lo que hace que YOLO sea más rápido respecto a otros sistemas de detección de objetos en tiempo real es que estos reutilizan clasificadores o localizadores para realizar la detección y aplican el modelo a una imagen en varias ubicaciones y escalas. Con YOLO, el enfoque es diferente. YOLO aplica una única red neuronal a la imagen completa. Esta red divide la imagen en regiones mediante una cuadrícula y predice los *bounding boxes* y las probabilidades para cada región. Estos *bounding boxes* están ponderados por las probabilidades predichas.

Las ventajas principales de YOLO respecto al resto de clasificadores es que mira la imagen completa una única vez en el momento de la prueba por lo que las predicciones están informadas por el contexto global de la imagen y además realiza predicciones usando solo una única red neuronal.

En este proyecto utilizamos YOLO reentrenado para que solo detecte las manos en las imágenes por lo que esta herramienta se utiliza para recortar la imagen ya que, como explicaremos más adelante, podemos saber en qué parte de la imagen está la mano por lo que al recortar la imagen en la zona donde está la mano eliminamos información adicional que puede perjudicar a la estimación.

4.4 Elementos Hardware

Uno de los principales problemas del *deep learning* es la necesidad de tener un equipo capaz de procesar el entrenamiento de la red así como tener una buena GPU con mucha memoria RAM para que los *batches* de imágenes que entren en la red sean los mayores posibles.

En este proyecto se ha utilizado un servidor con una GPU Geforce RTX 2080 con 11 gigas de RAM lo que hacía que los experimentos rondaran los 3 días de entrenamiento, excepto los últimos que duraban una semana.

4.5 Dataset

El dataset usado en este proyecto es (Gomez-Donoso y cols., 2019b) y está pensado para usarse en estimación de la pose de la mano 2D y 3D y para la locación de la mano. Está estructurado en diferentes secuencias y estas a su vez en grupos de frames. Cada uno de estos frames viene acompañado de un *ground truth* que incluye los puntos 3D de los *joints* de la mano que son dados por el *Leap Motion*. El *Leap Motion* es un aparato que captura de manera precisa la pose de la mano y da las posiciones 3D de los *joints* de las manos por lo que evitamos la necesidad de calcular estas posiciones de manera manual. Estos puntos representan cada falange de los dedos y la punta de cada uno, además de la posición de la

palma y de su normal. Las anotaciones del dataset serían las coordenadas X,Y,Z de los puntos que aparecen en la Figura 4.4 además del punto de la palma y de la normal de esta.

Figura 4.4: Puntos de las anotaciones del dataset

Las secuencias fueron capturadas con diferentes condiciones para asegurar una alta variabilidad, se cambió el sujeto, el momento del día y la velocidad de movimiento. Esta alta variabilidad ayudará a que el modelo pueda generalizar mejor cuando se utilice en el mundo real.

(a) Cámara brazo izquierdo

(b) Cámara brazo derecho

(c) Cámara central

(d) Cámara frontal

Figura 4.5: Los cuatro puntos de vista del dataset

Para poder tener un dataset equilibrado y que tuviese muestras de varias perspectivas se

realizó un aparato de captura de imágenes. Este aparato tiene una estructura de aluminio con tres brazos articulados que sostiene cuatro cámaras de las cuales tres son Loigtech C920 Pro y la otra, la que está en el brazo del medio y que captura la posición cenital, es una Microsoft LifeCam Studio. De las otras tres cámaras, dos de ellas están puestas en los brazos izquierdo y derecho con una orientación de 45º y la última está puesta en la base de la estructura. Por último, el *Leap Motion* está colocado en el cuadro de aluminio enfrente de las cámaras. Estas cámaras dan imágenes de gran calidad hasta 1080p aunque fueron capturadas con una resolución de 640x480. Como vemos en la Figura 4.5 esos serían los cuatro puntos de vista resultantes y las cuatro imágenes forman el dataset.

5 Estimación de la pose de la mano a partir de una imagen

5.1 Pipeline en general

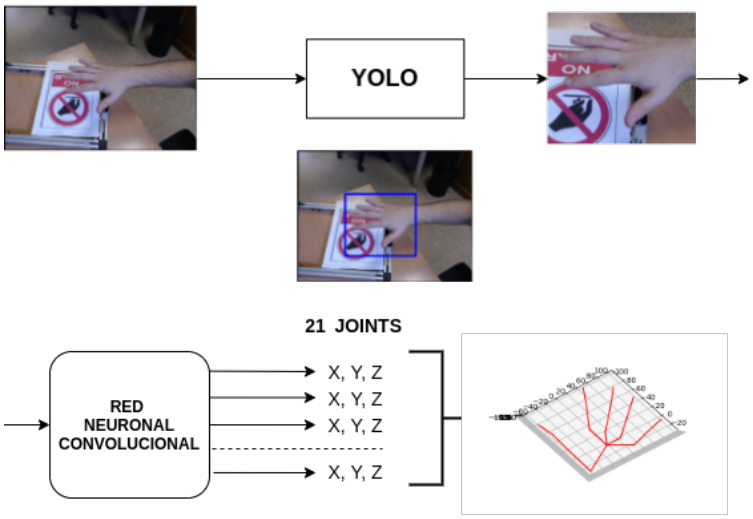


Figura 5.1: Pipeline

En la Figura 5.1 tenemos un vistazo general al funcionamiento completo del proyecto desde la captura de la imagen hasta la predicción puesta a modo de gráfica para representar la pose de la mano. Los pasos que siguen este pipeline son los siguientes:

- **Captura de la imagen:** usando cualquier cámara se irán capturando los frames uno a uno para procesarlos, dado que para crear los archivos hdf5 usamos un paquete de python que lee las imágenes en rgb y capturamos los frames de la cámara con OpenCV que los lee en bgr debemos hacer una transformación para que la red funcione correctamente.

- **Yolo:** una vez capturada la imagen le pasaremos la red de Yolo para que, como hemos explicado anteriormente, detecte la zona de la imagen donde se encuentre la mano, con el fin de eliminar zonas de la imagen que solo puedan confundir a la red a la hora de predecir y recortamos la imagen capturada con el *bounding box* detectado por Yolo.

- **Red Neuronal:** a la red neuronal le pasamos la imagen recortada y esta predice los

63 puntos correspondientes a los joints y nos devolverá un array con 63 datos, las X, Y y Z de los 21 *joints* correspondientes.

- **Representación final:** finalmente con los 63 datos predichos realizamos una representación en un gráfico de la pose de la mano, ya que los datos en el array predicho están en orden podemos saber cada 3 datos a que *joint* corresponde y así poder realizar la representación de la pose de la mano.

Una vez explicado el pipeline general, vamos a ir explicando qué se ha hecho para que este funcione, desde el procesamiento del dataset para hacer los archivos hdf5 para entrenar la red hasta el uso de las redes neuronales.

5.2 Preprocesamiento

Las anotaciones del dataset, es decir, los puntos de cada *joint* están relativos al *Leap Motion* con su propio sistema de referencia por lo que si se utilizara otra cámara para capturar los gestos, aunque un gesto fuese el mismo que el capturado por las cámaras del dataset no lo daría como tal ya que el sistema de referencia ha cambiado y capturar los gestos exactamente igual es imposible porque siempre hay pequeñas variaciones. Este problema se puede evitar cambiando el sistema de referencia y no utilizar las cámaras como referencia sino otro elemento.

Los gestos siempre van a venir definidos por las manos por lo que podemos utilizar la mano como sistema de referencia de forma que los puntos capturados siempre van a tener unas coordenadas parecidas en relación a la propia mano, independientemente de si la mano está más cerca de la cámara, más lejos, más a la izquierda, etc.

Figura 5.2: Vectores palma-índice y palma-pulgar

Por lo cual lo primero que hay que obtener es el nuevo sistema de referencia, unos nuevos ejes X, Y y Z que sean relativos a la mano. Para ello utilizamos tres puntos que definan el plano: el punto de la palma, el primer punto del índice ya que estos dos nunca van a cambiar

y el primer punto del dedo pulgar, que a pesar de que este sí que cambia vamos a explicar a continuación qué hemos hecho para solventar ese problema, y con estos tres puntos se crean dos vectores, uno que vaya de la palma al índice y otro de la palma al pulgar, como vemos en la Figura 5.2.

Después hacemos dos productos vectoriales entre los dos vectores que sacamos, y dado que el pulgar sí que se mueve, hacemos otro producto vectorial entre el anterior calculado y el vector palma-índice, y esos 3 vectores son nuestro nuevo sistema de referencia. Esos vectores quedarían de la manera que vemos en la Figura 5.3.

Figura 5.3: Vectores que forman el nuevo sistema de referencia

Por último, para transformar cada punto de un sistema de referencia a otro, utilizaremos una matriz de transformación, para ello calculamos los ángulos entre los dos sistemas de referencia con la fórmula que vemos en Ecuación 5.1, siendo A y B dos vectores. Con esta fórmula simplemente deberíamos sacar los ángulos entre el sistema de referencia de la cámara, que son los vectores que vemos en la Ecuación 5.2, y los vectores que hemos calculado antes.

$$\theta(A, B) = \arccos \frac{A \cdot B}{|A| \cdot |B|} \tag{5.1}$$

$$\begin{aligned} X_c &= [1 \;\; 0 \;\; 0] \\ Y_c &= [0 \;\; 1 \;\; 0] \\ Z_c &= [0 \;\; 0 \;\; 1] \end{aligned} \tag{5.2}$$

Después construimos esta matriz de transformación como vemos en la Ecuación 5.3, donde ax, ay y az son los ángulos de cada eje calculados antes. Una vez la tenemos, para transformar un punto simplemente tenemos que multiplicar esta matriz de transformación por la inversa

del punto y obtendremos el punto en el nuevo sistema de referencia.

$$T = \begin{bmatrix} \cos a_z \cdot \cos a_y & \cos a_z \cdot \sin a_y \cdot \sin a_x - \sin a_z \cdot \cos a_x & \sin a_z \cdot \sin a_x + \cos a_z \cdot \sin a_y \cdot \cos a_x & x \\ \sin a_z \cdot \cos a_y & \cos a_z \cdot \cos a_x + \sin a_z \cdot \sin a_y \cdot \sin a_x & \sin a_z \cdot \sin a_y \cdot \cos a_x - \cos a_z \cdot \sin a_x & y \\ -\sin a_z & \cos a_y \cdot \sin a_x & \cos a_y \cdot \cos a_x & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.3}$$

De esta forma, con cada gesto del dataset, calculamos la matriz de transformación a partir de los primeros puntos de la palma, índice y pulgar y la usamos para transformar todos los puntos de ese gesto, y con estos puntos transformados junto con las imágenes creamos los archivos hdf5 que los usamos para entrenar y probar la red neuronal convolucional. Este hdf5 contiene dos datasets distintos, por un lado tenemos el dataset que contiene todas las imágenes redimensionadas y por otro lado tenemos otro dataset que contiene las anotaciones de los *joints* correspondientes a cada imagen.

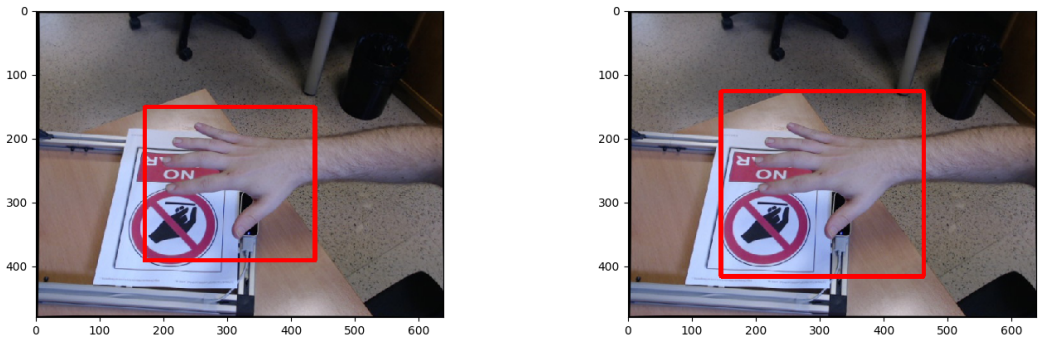
5.3 YOLO

Tras el primer experimento donde utilizamos las imágenes sin más, pensamos que la información adicional de la imagen que no correspondía con la mano podría reducir el rendimiento de la red. Por lo que decidimos utilizar YOLO para reducir la información de la imagen y así solamente meter la parte de la imagen donde se encuentra la mano.

Como hemos visto YOLO nos devuelve los puntos del *bounding box* donde se encuentra los objetos de la imagen. En este caso, dado que YOLO lo tenemos entrenado para que detecte las manos en las imágenes, primero pasamos YOLO y recortamos la imagen con el *bounding box* devuelto, para después meter las imágenes en los archivos hdf5 junto con los puntos transformados. Cuando probamos la red en tiempo real, a cada frame que capturamos de la webcam hacemos lo mismo, la recortamos y una vez hecho esto se la pasamos a la red neuronal convolucional.

Haciendo experimentos descubrimos que uno de los problemas de usar YOLO es que en muchas ocasiones el *bounding box* que devolvía dejaba parte de la mano fuera. Para evitar esta situación, que ocurría a veces, al *bounding box* le añadimos a cada lado 25 píxeles (este valor se ha calculado de manera empírica) de manera que a las imágenes que no les pasaba eso no se añade tanta información inservible y a las que le pasaba con este añadido las manos ya no quedan fuera, como vemos en la Figura 5.4. Esto se lo hacemos tanto a las imágenes del dataset para recortarlas con ese nuevo *bounding box* para meterlas en los archivos hdf5 como a los frames capturados por la webcam.

Otras de las cosas que descubrimos haciendo experimentos es que YOLO estaba entrenado con pocas posiciones de la mano por lo que si la giras un poco en alguna dirección YOLO no detecta el *bounding box* y sin esto la imagen no entra a la red para predecir. Dos ejemplos de



(a) YOLO sin aumentado de píxeles **(b)** YOLO con aumentado de píxeles

Figura 5.4: Ejemplos de YOLO

posiciones que no capta YOLO las podemos ver en la Figura 5.5 donde vemos que el cuadrado no aparece donde la mano, si no en la esquina de arriba a la izquierda que es el sitio puesto para que aparezca cuando no detecta la mano.

(a) Ejemplo 1 **(b)** Ejemplo 2

Figura 5.5: Ejemplos de YOLO no encontrando las manos en la imagen

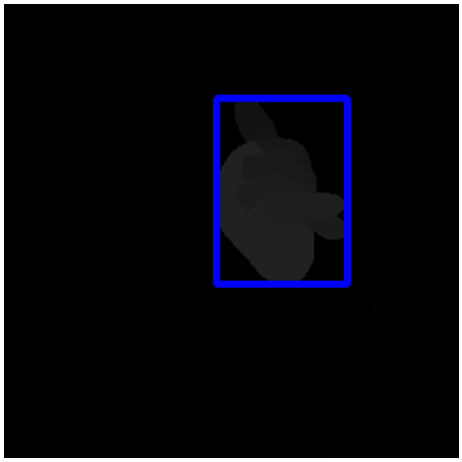
Para solucionar este problema hemos decidido reentrenar YOLO. El dataset que hemos utilizado para reentrenar YOLO está formado de la siguiente forma:

- 1000 imágenes del dataset utilizado en la estimación de la pose de mano cuya anotación nos la ha dado el propio YOLO, aplicando el formato que necesita YOLO en sus anotaciones:
  <clase> <x de la mitad del *bounding box* / ancho imagen> <y de la mitad del *bounding box* / alto imagen> <ancho / ancho imagen> <alto / alto imagen>
- 1000 imágenes generadas a partir de las 1000 anteriores aplicándoles rotaciones aleatorias a cada una de ellas. Las anotaciones las hemos sacado de manera manual con un programa en python con OpenCV, sacamos el *bounding box* clicando en la esqui-

na superior izquierda y derecha de este, pasamos esos valores a la función que hemos mencionado antes y generando las anotaciones una a una.

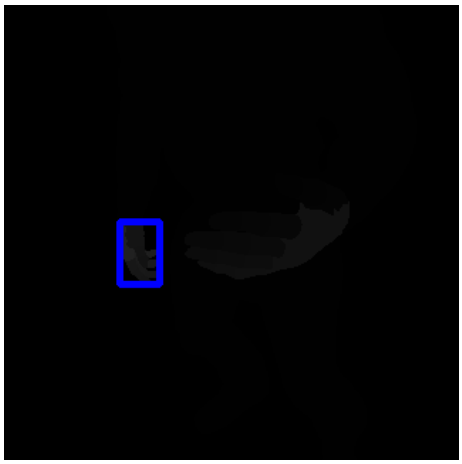
Después de entrenarlo y probarlo, vimos que con el dataset funcionaba bien pero al probarlo con una webcam en vivo no, por lo que existía un problema de generalización que lo solventamos añadiendo imágenes de otros datasets.

(a) Ejemplo 1 de imagen de color



(b) Ejemplo 1 de la máscara

(c) Ejemplo 2 de imagen de color

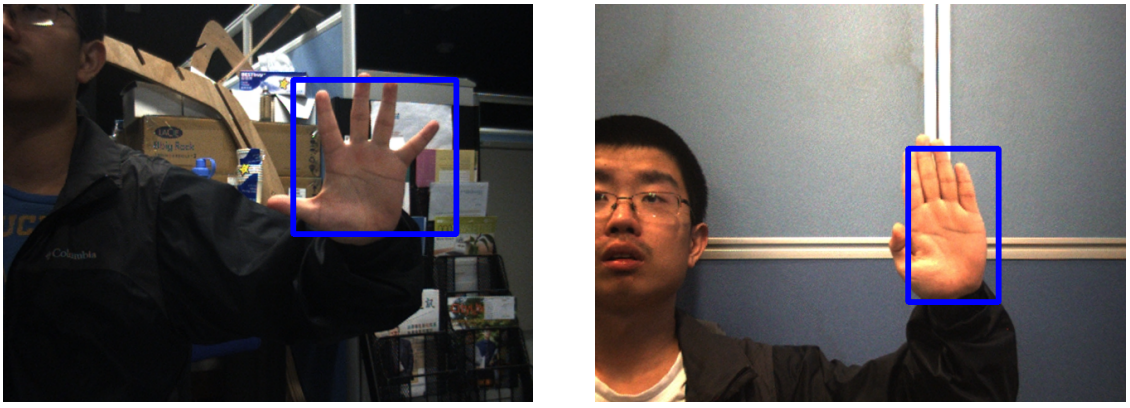


(d) Ejemplo 2 de la máscara

Figura 5.6: Ejemplos de imágenes con la anotación del dataset Rendered Handpose

Usamos 2000 imágenes del dataset *Rendered Handpose* (Zimmermann y Brox, 2017a). Este dataset está formado por imágenes hechas en blender y nos da las imágenes, el mapa de profundidad y la segmentación de la mano. Las anotaciones las hicimos de manera automática, obteniendo los puntos del *bounding box* y pasándoselos a la función descrita anteriormente de la siguiente forma: recorremos todos los píxeles de la segmentación y guardamos las coordenadas de los píxeles que tengan un valor de máscara correspondiente a la mano derecha, y después cogemos la mayor y menor X e Y, obteniendo el *bounding box* (Figura 5.6).

También intentamos usar 2000 imágenes del dataset *Stereo Hand Tracking* (Zhang y cols., 2016). En este dataset hay seis secuencias de manos izquierdas, cada secuencia con un fondo diferente, por lo que como estamos buscando que generalice mejor usamos imágenes de las seis secuencias. Al ser manos izquierdas deberíamos haberles hecho un *flip* tanto a estas imágenes como al *bounding box* y una vez hecho esto, pasarle a la función las nuevas coordenadas del *bounding box* para que nos saque la anotación que necesita YOLO para entrenar. El problema que tenemos es que las anotaciones no están bien calibradas con las imágenes, y el *bounding box*, como vemos en la Figura 5.7 que obtenemos usando las menores y mayores X e Y, las cuales sacamos transformando los puntos 3D a 2D multiplicando los puntos 3D por la matriz de parámetros intrínsecos, no corresponde con la imagen. Aparte intentamos ajustarlo lo máximo posible, multiplicando por 0.001 y por un número que dependiendo de la secuencia que sea cambiaba, y aún así en muchas imágenes se deja parte de la mano fuera, por lo que no podemos utilizarlo para mejorar el entrenamiento de YOLO.

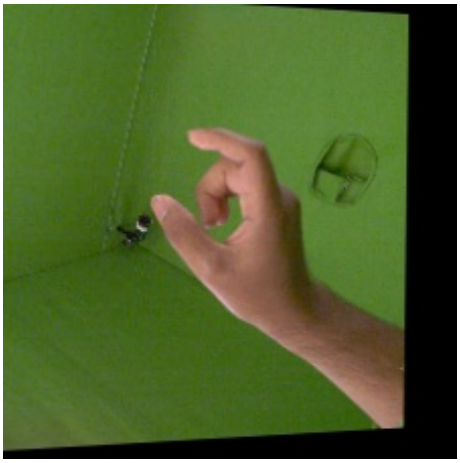


(a) Ejemplo 1 **(b)** Ejemplo 2

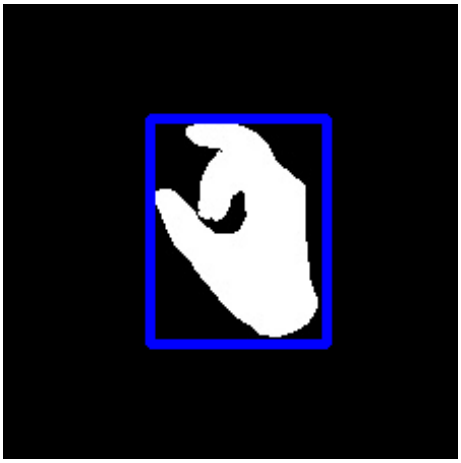
Figura 5.7: Ejemplos de imágenes con la anotación del dataset Stereo Hand Tracking

Dado que buscábamos un dataset en el que las manos fuesen reales, ya que nuestro entorno detecta manos reales por lo que cuanto más ejemplos de estas mejor iba a funcionar, y que tuviese anotaciones o máscara para sacar las anotaciones que necesita YOLO de manera automática además de una gran variabilidad de poses, encontramos el dataset FreiHand (Christian Zimmermann y Brox, 2019). Este dataset contiene imágenes a color así como su máscara, de estas imágenes cogimos 2000 al azar e hicimos el mismo proceso que hemos descrito anteriormente, buscamos las coordenadas la máscara y usando la máxima y mínima X e Y sacamos el *bounding box* y con estos cuatro puntos se lo pasamos a la función que nos da las anotaciones que necesita YOLO para entrenar (Figura 5.8).

Después de entrenar YOLO descubrimos que las perspectivas de las manos que antes no detectaba, ahora sí que las detectaba, como observamos en la Figura 5.9. Pero encontramos otro problema: dado que el último dataset utilizado tenía muchas imágenes de interacción con objetos es posible que haya afectado a que en gestos que tengan algunos dedos levantados el *bounding box* deje parte de la mano fuera como podemos ver en la Figura 5.9.

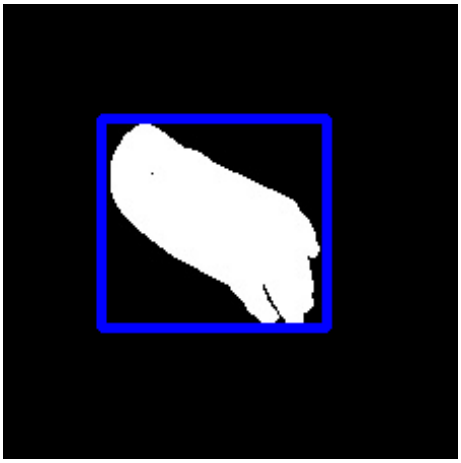


(a) Ejemplo 1 de imagen de color



(b) Ejemplo 1 de la máscara

(c) Ejemplo 2 de imagen de color



(d) Ejemplo 2 de la máscara

Figura 5.8: Ejemplos de imágenes con la anotación del dataset Freihand

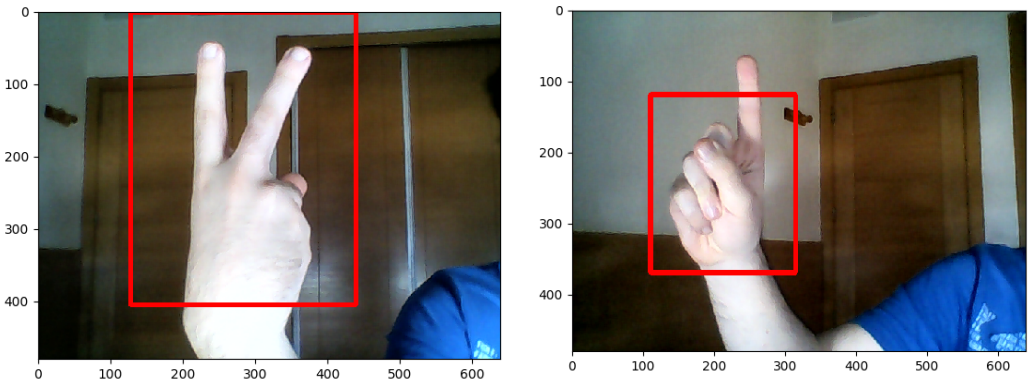


Figura 5.9: Izquierda: funcionamiento correcto de pose antes no detectada. Derecha: Problema encontrado en ocasiones

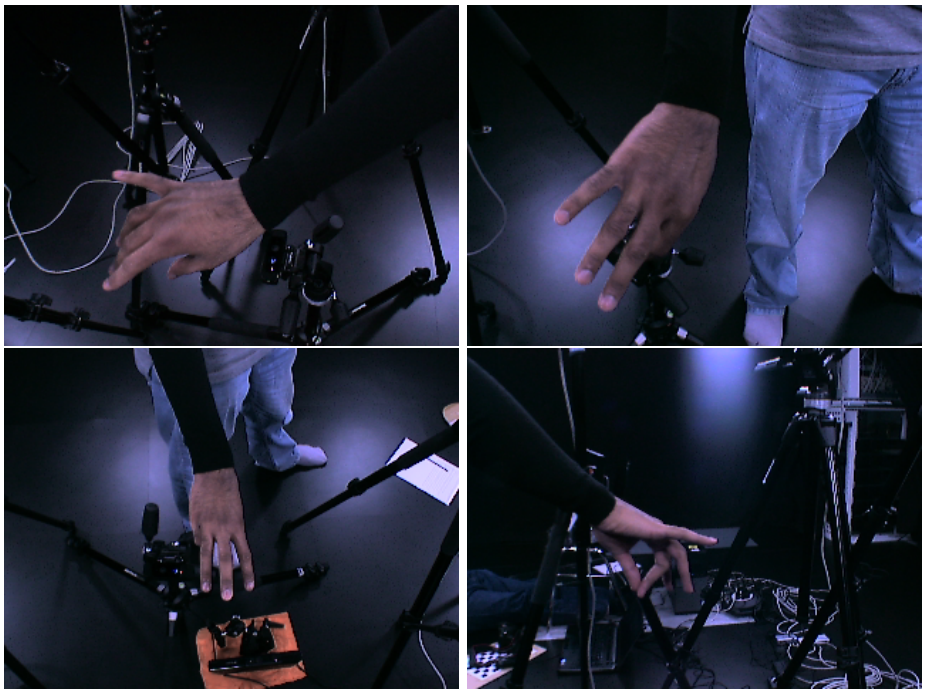


Figura 5.10: Ejemplos de imágenes del dataset Dexter

Figura 5.11: Ejemplos del correcto funcionamiento de YOLO

Para terminar de entrenarlo bien, encontramos el dataset Dexter, (Sridhar y cols., 2013). En este dataset hay una secuencia de imágenes que va cambiando las posiciones de los dedos, cerrando algunos, abriendo otros, cosa que necesitábamos ya que teníamos el problema de que nos dejaba los dedos fuera. Además este dataset dispone de diferentes cámaras para cada gesto, por lo que tiene una gran variabilidad y nos puede ayudar a la hora de cambiar la cámara de perspectiva (Figura 5.10). Cogimos 1960 imágenes y para las anotaciones de YOLO lo hemos hecho de manera manual como con las 1000 imágenes generadas con *data*

augmentation.

Con este último entrenamiento, conseguimos corregir el problema del entrenamiento anterior y ya no solo nos detectaba la mano tanto enfocando a la palma como al dorso sino que además conseguimos que no dejara ningún dedo fuera como vemos en los ejemplos en la Figura 5.11. Además conseguimos robustez, ya que como vemos en Figura 5.11 funciona con diferentes tonos de luz, natural y artificial y con distintas poses y perspectivas de la mano.

5.4 Red neuronal convolucional

Para predecir la pose de la mano 3D hemos utilizado una red neuronal convolucional. En este caso para realizar los experimentos hemos utilizado tres arquitecturas diferentes como hemos todas ellas de Keras como hemos mencionados en 4.2:

- **ResNet50**
- **InceptionV3**
- **DenseNet**

Lo que hacemos con todas estas redes es quitarle la última capa poniendo el argumento *include_top* a *False* de manera que podemos poner nosotros la última capa que nos interese, de manera que pusimos una capa con 63 neuronas que corresponden a los 63 puntos (21 *joints* con componentes X, Y y Z) que queremos predecir.

Otra cosa que hacíamos con las redes neuronales convolucionales es que la función de pérdida se la pasábamos nosotros, es decir, no utilizábamos una que estuviera predefinida por Keras. Esta función de error era la raíz del error cuadrático medio la cual representa la raíz cuadrada del segundo momento de la muestra de las diferencias entre los valores previstos y los valores observados o la media cuadrática de estas diferencias. Como observamos en la Ecuación 5.4.

$$\sqrt{\frac{\sum_{t=1}^{T}(\hat{y}_t - y_t)^2}{T}} \tag{5.4}$$

5.5 Clusters

Otro de los problemas que nos encontramos cuando probamos los modelos en vivo, es decir, capturando los frames de una webcam, fue ver que solo predecía bien los frames en los que hubiera una mano abierta o cerrada del todo, por lo que pensamos que tal vez las imágenes del dataset estuvieran descompensadas, que hubieran muchas más imágenes de manos abiertas o cerradas que con otros gestos. Al no tener las imágenes agrupadas en clases decidimos realizar un proceso de clusterización para agrupar las imágenes con gestos "parecidos" para conocer cuál era la magnitud de la descompensación.

Para hacer esta clusterización lo que hicimos fue ver qué imágenes se parecían dentro de un umbral. Teníamos una lista de listas y por cada imagen del dataset la comparábamos con la primera imagen de cada lista, si se daba el caso que alguna estuviese por debajo del umbral seleccionado se añadía a esa lista y si había varias con un valor menor de ese umbral, se añadía a la lista con menor valor. En el caso de que con ninguna imagen diese un valor por debajo del umbral se añadía una nueva lista a la lista de listas.

Para calcular ese ”parecido” entre gestos usamos la distancia Euclídea entre las manos. Para ello calculamos la distancia Euclídea entre los 21 puntos correspondientes a los *joints* de la mano, los sumamos todos y los dividimos entre 21 y con este valor es el que usamos para comparar con el umbral previamente dicho.

Una vez calculadas todas las imágenes del dataset, la lista de listas la pasamos a un gráfico de barras, donde la x era cada *cluster* y la y era la cantidad de imágenes. Además, para después operar con esas imágenes nos guardamos esa lista de listas serializándola en un archivo.

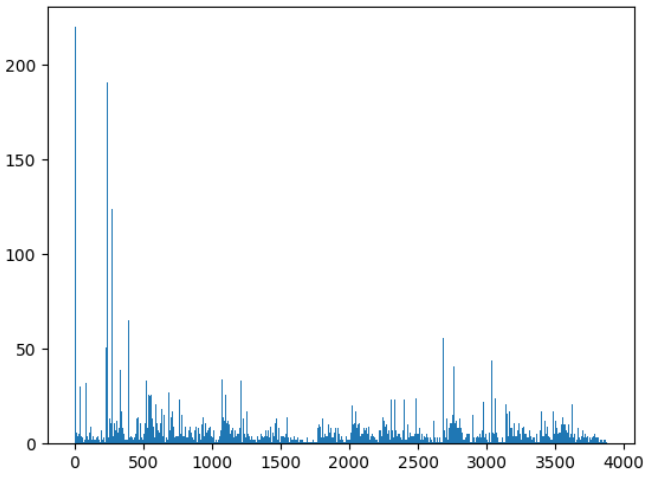


Figura 5.12: Gráfica de los clusters

Como vemos en la Figura 5.12 hay una gran descompensación entre clases y al ordenar las listas de mayor a menos podemos ver en la Figura 5.13 que el gesto más repetido es la mano abierta y luego hay varios con clusters con una imagen solo. El primer experimento que hicimos para compensar este desbalanceo fue cortar cada *cluster* a 50 imágenes y entrenar con esas imágenes, pero como hemos visto había algunos que solo tenían una imagen o dos por lo que esto no fue viable y nos dispusimos a hacer *data augmentation* que se explica en el siguiente apartado.

(a) Ejemplo de imagen con gesto más repetido

(b) Ejemplo de imagen con gesto menos repetido

Figura 5.13: Ejemplos de imágenes de menor y mayor repetición

5.6 Data Augmentation

Después de hacer el experimento e cortar todos los clusters por el mismo número vimos que no iba bien por lo que en vez de reducir imágenes decidimos aumentarlas haciendo data augmentation con el siguiente proceso: Cada cluster tendría que acabar con 200 imágenes, para lo cual los clusters que tuviesen 50 imágenes o más no tendrían data augmentation ya que cada imagen en verdad son 4, ya que son 4 cámaras. Los clusters que tienen menos de 50 imágenes, irían haciendo data augmentation de cada imagen hasta llegar a 200, es decir, primero cargaríamos en una lista las imágenes del dataset correspondientes a ese cluster y con cada una generaríamos otra imagen haciendo data augmentation hasta llegar a 200. En los casos que hubiesen tan pocas imágenes en los cluster que generando una imagen por cada imagen del cluster no llegasen a 200, se iría generando de manera exponencial, es decir, se generarían primero a partir de las imágenes del cluster, y una vez que no queden imágenes en el cluster, empezaríamos a generar imágenes a partir de las nuevas imágenes que hemos generado antes, de manera que ninguna imagen sería exactamente igual a las otras.

Para asegurar que las imágenes fuesen lo más diferentes posibles realizamos cuatro cambios: rotaciones de -45 grados a 45 grados, difuminado, ruido gaussiano y cambio de contraste lineal, que son los cambios que vemos en la Figura 5.14. Además, estos cambios se hacen en mayor o menor medida de manera aleatoria y aparte, en cada imagen generada, se aplica de uno a cuatro cambios de manera aleatoria haciendo que toda imagen generada sea prácticamente imposible que sea igual a alguna ya existente.

Otro *data augmentation* que hicimos fue respecto a las imágenes de la cámara frontal, ya que las imágenes de este tipo estaban descompensadas respecto a las otras, por lo que generamos 3 imágenes por cada una, para igualar con el resto de imágenes del dataset.

(a) Rotación

(b) Difuminado

(c) Cambio contraste

(d) Ruido Gaussiano

Figura 5.14: Cambios DA

6 Resultados

6.1 Primeros experimentos

El primer experimento que hicimos fue entrenar la red, cuya arquitectura era una ResNet50, con las imágenes sin tratar, es decir, sin aplicarles YOLO. Después de generar el dataset redimensionando las imágenes a 224x224 y haciendo el preprocesamiento de los *joints* ya explicado, entrenamos la red obteniendo los resultados que vemos en la Figura 6.1. Cuando terminábamos un experimento cogíamos los pesos de la época con menos *loss* y calculábamos el error medio en el conjunto de test. Este error nos lo daba la ya mencionada distancia Euclídea, que como hemos comentado se calculaba y se sumaba la distancia Euclídea de cada *joint* entre lo previsto por la red y el *ground truth*, y se dividía entre el número de *joints*. Una vez que teníamos este valor de todo el conjunto de test se dividía por el número de elementos para sacar la media. En este caso era de **3.599**, que significaría que hay un error de 3.599 milímetros ya que las anotaciones están en esta medida. Sin embargo, esto era así con el dataset ya que al utilizarlo en un entorno real con una webcam no funcionaba bien.

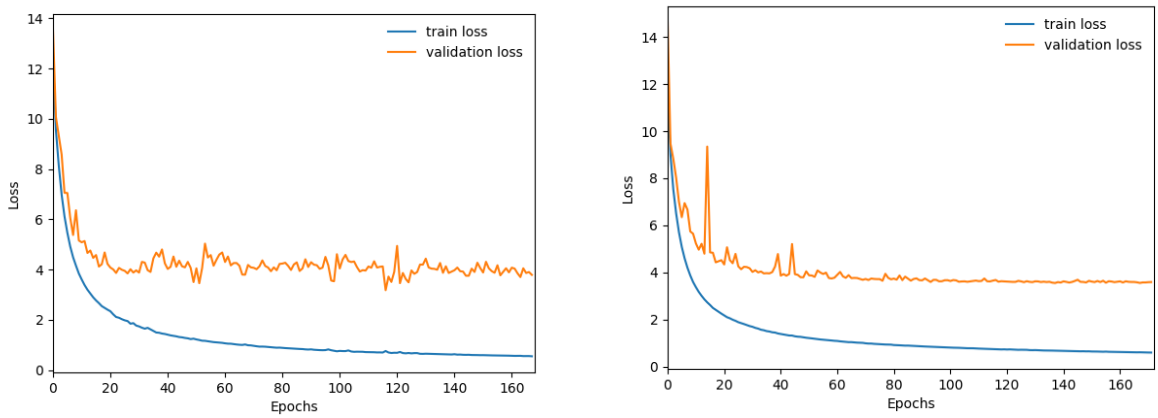


Figura 6.1: Gráficas loss train-validation primeros experimentos. Izquierda: sin Yolo. Derecha: con Yolo

Una vez hecho este experimento pensamos que una de las causas por las que no funcionaba bien al utilizarlo en un entorno real es que a lo mejor la red estaba aprendiendo información adicional de las imágenes del dataset por lo que decidimos utilizar YOLO que como ya hemos dicho estaba entrenado para reconocer las manos. Entonces el dataset de entrenamiento, de validación y de test estaba formado por las imágenes cortadas por el *bounding box* de YOLO. Utilizamos una ResNet50 y obtuvimos los resultados que vemos en la Figura 6.1. Como en

el anterior experimento y como haremos en todos, calcularemos el error medio del conjunto de test que en este caso es **4.607**, como vemos es mayor que el anterior, lo que no debería ser así pero que descubrimos que en algunas ocasiones YOLO dejaba parte de la mano fuera por lo que a la red puede ser que le faltase información. Esto lo solucionamos más adelante aumentando la cantidad de píxeles a cada lado del *bounding box*. Además, seguía sin funcionar en un entorno real.

6.2 Experimentos con Cluster

Cuando descubrimos que las imágenes estaban desbalanceadas en cuanto a las poses, hicimos el proceso de clustering explicado en el apartado de Desarrollo. El primer experimento que hicimos con este clustering fue cortar todos los clusters a 50 imágenes por cluster (200 imágenes ya que son 4 cámaras). Los resultados obtenidos, utilizando una ResNet50 en el entrenamiento lo vemos en la Figura6.2 y en el conjunto de test obtuvimos **3.641**. Sin embargo, esto no funcionaba nada bien en un entorno real, ya que aunque habíamos reducido el número de los gestos más repetidos seguíamos teniendo el problema de que algunos clusters tenían muy pocas imágenes, y además, ahora habíamos reducido el número de imágenes total. También cabe destacar que en este experimento YOLO seguía dejando en algunas imágenes parte de la mano fuera.

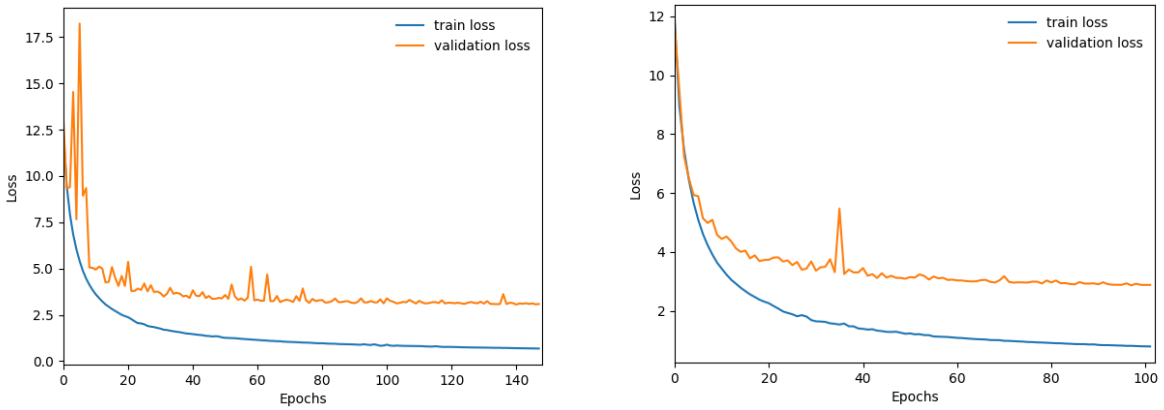


Figura 6.2: Gráficas loss train-validation de los experimentos con clusters. Izquierda: sin aumentado de píxeles en YoOLO. Derecha: con aumentado de píxeles

Para solucionar el problema de YOLO dejando fuera parte de la mano decidimos aumentar el *bounding box* que devolvía en 25 píxeles a cada lado, de esta manera las imágenes que dejaba fuera parte de la mano ya no la dejaba fuera y a las que no las dejaba fuera solo aumentaban un poco y no cogían demasiada información adicional innecesaria, como hemos visto en la explicación de esto. Para ver si esto afectaba de verdad o no, hicimos un experimento usando los clusters y aumentando este *bounding box* utilizando una ResNet50, como vemos en la Figura 6.2. El entrenamiento mejoró un poco y en el error medio que obtuvimos **3.427** vemos que mejora, no solo al experimento del cluster sin aumentado sino que también a los dos

anteriores experimentos, por lo que el aumentado de píxeles lo realizaremos también a partir ahora en todos los experimentos siguientes.

6.3 Experimentos con distintas arquitecturas

Otro de los experimentos que quisimos realizar fue ver si alguna de las arquitecturas de Keras iba mejor que la que estábamos usando en todos los experimentos anteriores, la ResNet50. Decidimos probar dos arquitecturas diferentes: Inception y DenseNet. El dataset que hemos utilizado para realizar estos experimentos ha sido el mismo que el anterior, los clusters cortados a 50 y añadiendo los 25 píxeles al *bounding box* dado por YOLO. En la Figura 6.3 tenemos los resultados de entrenamiento utilizando la Inception. Como observamos ha bajado bastante el *loss* de validación, y no solo eso, sino que el error medio en el conjunto de test ha bajado muchísimo, llegando a un error de **2.89**, siendo el experimento que mejor valor hemos tenido hasta ahora en el conjunto de test. Sin embargo, seguimos teniendo un problema a la hora de probarlo en un entorno real, y es que solo funciona con los gestos de mano abierta y mano cerrada ya que seguimos teniendo pocas imágenes de otros gestos.

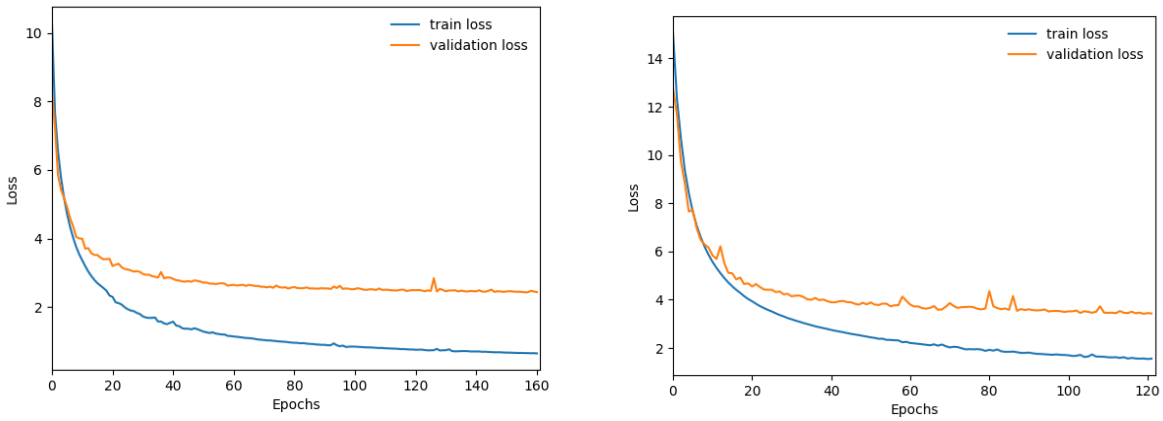


Figura 6.3: Gráficas loss train-validation de los experimentos con arquitecturas. Izquierda: Inception. Derecha: DenseNet

Viendo la clara mejora que obtuvimos cambiando la arquitectura a la Inception, decidimos hacer un experimento probando la DenseNet, una arquitectura más grande que la ResNet y la Inception. Por ello, también decidimos subir la resolución de las imágenes de 224x224 a 300x300. Sin embargo, en este caso no ocurrió lo mismo que con la prueba de la Inception como vemos en la Figura 6.3. En el caso del conjunto de test sí que bajó un poco con respecto a los primeros experimentos con las ResNet50 ya que obtuvimos un error medio de **2.994**. Sin embargo, utilizando el mismo dataset y el aumentado de píxeles de YOLO es la que peor resultados obtiene. Además de seguir sin funcionar en un entorno real.

Como conclusión a estos experimentos cambiando las arquitecturas, decidimos usar la Inception en los siguientes experimentos, viendo que había mejorado de manera notable los

Arquitectura	Validation loss	Test loss
ResNet50	2.86	3.427
Inception	**2.429**	**2.89**
DenseNet	3.41	2.994

Tabla 6.1: Comparativa de arquitecturas

resultados. Sin embargo, seguimos teniendo el problema de la no generalización, por ello, aprovechando el proceso de clusterización que hicimos, decidimos realizar *data augmentation*.

6.4 Experimentos de Data Augmentation

Dado que todas las pruebas que hicimos nos daban buenos resultados con el propio dataset pero que al probarlo en el entorno real solo predecía bien la pose de la mano abierta y mano cerrada decidimos que como teníamos el proceso de clusterización hecho, podría aumentar los ejemplos de aquellos clusters que tuviesen muy pocas imágenes como hemos visto en la Sección 5.6. Viendo el éxito y la mejora de la arquitectura Inception frente a las demás decidimos usar esta para este experimento y obtuvimos los resultados de entrenamiento que vemos en la Figura 6.4.

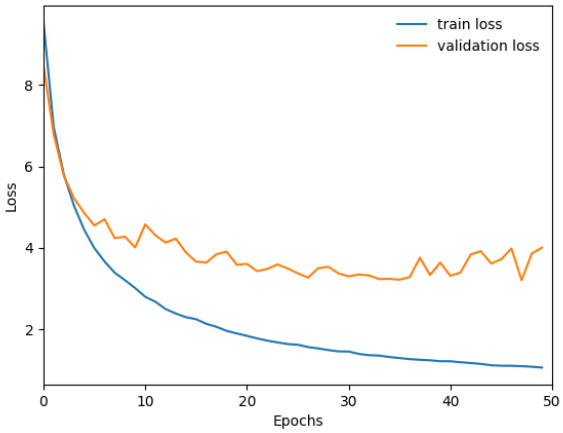
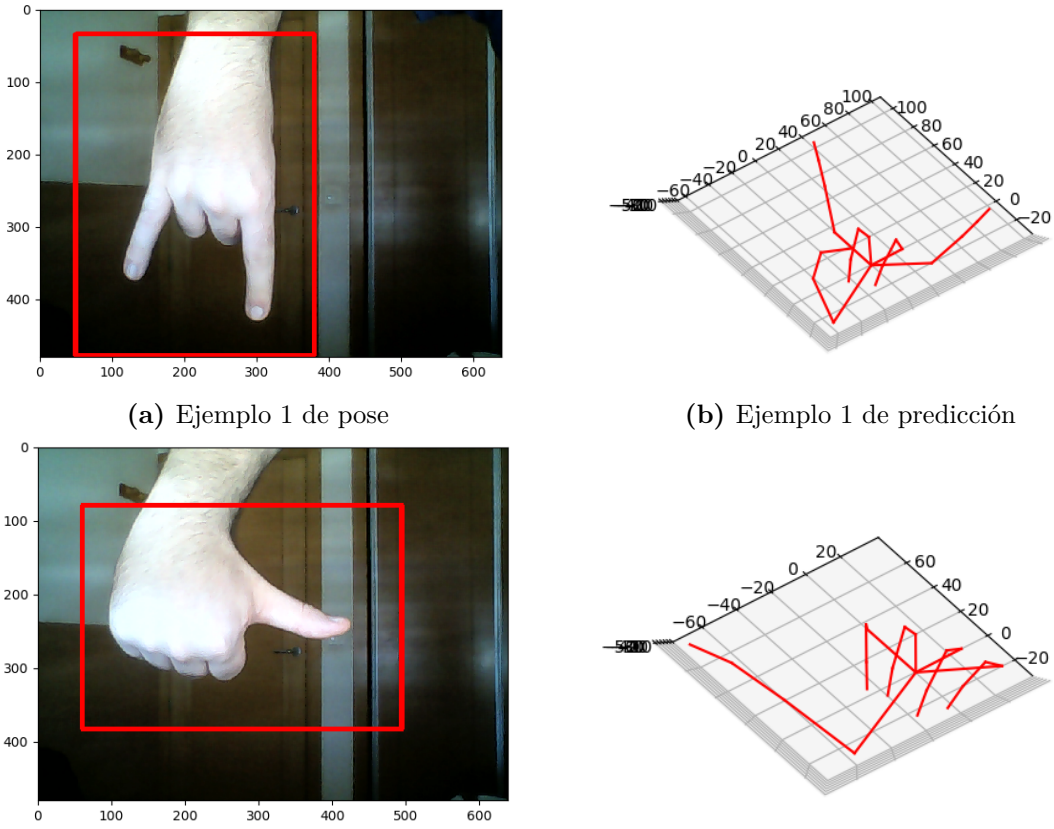


Figura 6.4: Gráficas loss train-validation del experimento con data augmentation

En cuanto al error del test obtuvimos un **4.649**. A simple vista puede parecer que el experimento es peor que los anteriores, puesto que el error es superior. Sin embargo, esto se debe a que ahora detecta la mayoría de los gestos y además el aumento de error es de solo 1 mm. Además este experimento nos sirvió para que funcionara en el entorno real, al menos desde una perspectiva de la mano. Como vemos en la Figura 6.5 con la mano en esa perspectiva sí que vemos como predice correctamente la pose.

Sin embargo, esa posición de la mano no es la ”natural” de una webcam, esta posición



(a) Ejemplo 1 de pose

(b) Ejemplo 1 de predicción

(c) Ejemplo 2 de pose

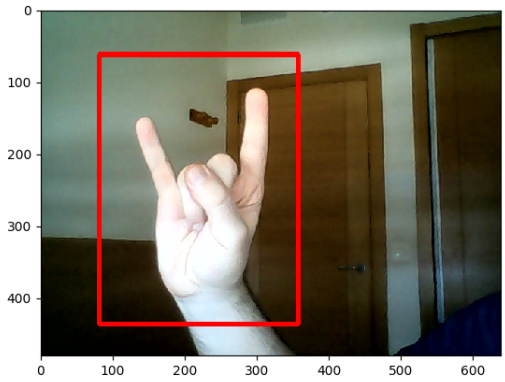
(d) Ejemplo 2 de predicción

Figura 6.5: Ejemplos de poses desde la perspectiva que funciona

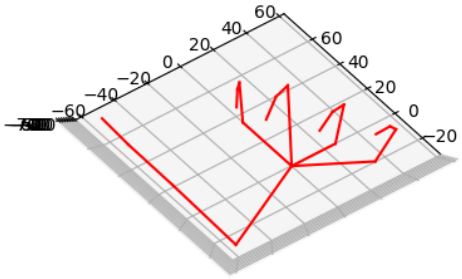
"natural" sería de frente y como vemos en la Figura 6.6, siendo la misma pose que en la Figura 6.5, no predice bien la pose desde esa perspectiva. Esto puede ser debido a que solo un cuarto de las imágenes del dataset son de frente por lo que al *data augmentation* hecho, decidimos que deberíamos generar tres veces más imágenes de esa vista para igualar a las del resto del dataset.

Con este nuevo dataset decidimos entrenar para intentar eliminar el problema de la perspectiva frontal que como hemos visto no funciona correctamente. Los resultados que obtuvimos en el entrenamiento los vemos en la Figura 6.7 y que obtuvimos un valor de **3.712** como error en el test. Como vemos, estos valores han reducido bastante los del experimento anterior.

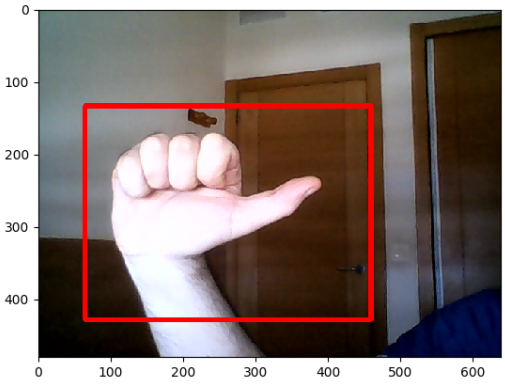
Sin embargo, esto no cambió nada, ya que la mayoría de imágenes del dataset desde la cámara frontal, la mano está inclinada no esta recta, por lo que no tiene ejemplos de este tipo haciendo muy difícil que pueda predecir correctamente desde esa perspectiva.



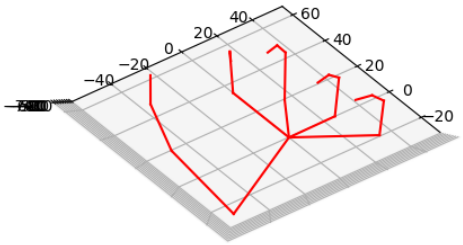
(a) Ejemplo 1 de pose



(b) Ejemplo 1 de predicción



(c) Ejemplo 2 de pose



(d) Ejemplo 2 de predicción

Figura 6.6: Ejemplos de poses desde la perspectiva que no funciona

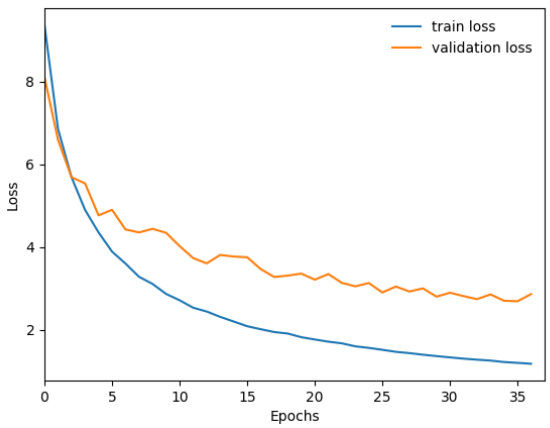


Figura 6.7: Gráficas loss train-validation del experimento con *data augmentation* en la perspectiva frontal.

7 Conclusiones

Se ha desarrollado un sistema que a partir de una imagen en 2D, nos predice la la pose de la mano en 3D, siendo estas predicciones los 21 *joints* de la mano. Además se ha desarrollado de manera que puede ser utilizado en tiempo real.

Como hemos visto en el apartado 6 con cada experimento que se ha ido haciendo se ha mejorado el método, pasando de no funcionar fuera del dataset hasta funcionar con diferentes cámaras. La principal ventaja de este método es que no necesitas una cámara con grandes prestaciones para que funcione. Hemos probado este método con dos tipos de cámara, con una webcam de un portátil y con la cámara de un móvil, siendo de diferentes calidades ambas nos han dado resultados similares, aunque si bien es cierto que cuanta más calidad tenga la cámara mejor irá. Así como la cámara, las pruebas realizadas sobre el funcionamiento son en un portátil con unas prestaciones bastante bajas, por lo que no es necesario un gran equipo para que funcione.

Una de las limitaciones que tiene el método es que la cámara tiene que estar en posición cenital, solo hemos obtenido buenas predicciones si la cámara está en esa posición, desde otras vistas las predicciones no son nada buenas. Como hemos comentado, este problema puede ser debido a que en el dataset de entrenamiento la mayoría de imágenes son de este tipo. Una posible solución a este problema sería mezclar diferentes datasets cuyos ejemplos no tengan esa posición cenital de la cámara.

Los posibles trabajos futuros que se podrían hacer para mejorar el método serían los siguientes:

- Utilizar Recurrent Neural Network (RNN), estas redes neuronales permiten que la información persista durante algunos pasos ó épocas de la etapa entrenamiento. Las conexiones entre nodos forman un gráfico dirigido a lo largo de una secuencia temporal. Esto podría ayudar debido a que los cambios entre gestos no son instantáneos sino que interviene el tiempo.

- Utilizar *pixel-wise* para segmentar la mano y así eliminar todo el fondo, esto ayudaría a mejorar la generalización debido a que solo importaría la mano. La red no podría memorizar el fondo lo que mejoraría por el hecho de utilizarlo en cualquier escenario.

El modelo final lo hemos aplicado a un brazo robótico de bajo coste, que es el trabajo de fin de grado de otro estudiante. Nuestro trabajo predice la pose de la mano y el brazo robótico utiliza esta predicción para imitar el movimiento, como podemos ver en este vídeo[1].

[1] `https://www.youtube.com/watch?v=PO08ff4NIXg`

Como resultado de este trabajo se ha enviado un artículo al congreso Workshop de agentes físicos 2020.

Bibliografía

Bilbeisi, G. (2019, 03). *3d hand pose estimation using depth rgb and rgbd images.*

Christian Zimmermann, J. Y. B. R. M. A., Duygu Ceylan, y Brox, T. (2019). Freihand: A dataset for markerless capture of hand pose and shape from single rgb images. En *Ieee international conference on computer vision (iccv).* Descargado de `https://lmb.informatik.uni-freiburg.de/projects/freihand/`

de La Gorce, M., Fleet, D. J., y Paragios, N. (2011). Model-based 3d hand pose estimation from monocular video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *33*(9), 1793-1805.

Ge, L., Ren, Z., Li, Y., Xue, Z., Wang, Y., Cai, J., y Yuan, J. (2019). *3d hand shape and pose estimation from a single rgb image.*

Gomez-Donoso, F., Orts-Escolano, S., y Cazorla, M. (2018). Robust hand pose regression using convolutional neural networks. En A. Ollero, A. Sanfeliu, L. Montano, N. Lau, y C. Cardeira (Eds.), *Robot 2017: Third iberian robotics conference: Volume 1* (pp. 591–602). Cham: Springer International Publishing. Descargado de `https://doi.org/10.1007/978-3-319-70833-1_48` doi: 10.1007/978-3-319-70833-1_48

Gomez-Donoso, F., Orts-Escolano, S., y Cazorla, M. (2019a). Accurate and efficient 3d hand pose regression for robot hand teleoperation using a monocular rgb camera. *Expert Systems with Applications*, *136*, 327 - 337. Descargado de `http://www.sciencedirect.com/science/article/pii/S0957417419304634` doi: https://doi.org/10.1016/j.eswa.2019.06.055

Gomez-Donoso, F., Orts-Escolano, S., y Cazorla, M. (2019b). Large-scale multiview 3d hand pose dataset. *Image and Vision Computing*, *81*, 25-33.

Grzejszczak, T., Kawulok, M., y Galuszka, A. (2016, diciembre). Hand landmarks detection and localization in color images. *Multimedia Tools Appl.*, *75*(23), 16363–16387. Descargado de `https://doi.org/10.1007/s11042-015-2934-5` doi: 10.1007/s11042-015-2934-5

Kuznetsova, A., Leal-Taixé, L., y Rosenhahn, B. (2013). Real-time sign language recognition using a consumer depth camera. En *2013 ieee international conference on computer vision workshops* (p. 83-90).

Marcel, S., y Bernier, O. (1999). Hand posture recognition in a body-face centered space. En A. Braffort, R. Gherbi, S. Gibet, D. Teil, y J. Richardson (Eds.), *Gesture-based communication in human-computer interaction* (pp. 97–100). Berlin, Heidelberg: Springer Berlin Heidelberg.

Marcel, S., Bernier, O., Viallet, J.-E., y Collobert, D. (2000). Hand gesture recognition using input-output hidden markov models. En *Proceedings of the fourth ieee international conference on automatic face and gesture recognition 2000* (p. 456). USA: IEEE Computer Society.

Molina, J., Pajuelo, J. A., Escudero-Viñolo, M., Bescós, J., y Sanchez, J. M. M. (2013). A natural and synthetic corpus for benchmarking of hand gesture recognition systems. *Machine Vision and Applications*, *25*, 943-954.

Oberweger, M., Wohlhart, P., y Lepetit, V. (2015). Training a feedback loop for hand pose estimation. En *2015 ieee international conference on computer vision (iccv)* (p. 3316-3324).

Panteleris, P., y Argyros, A. A. (2017). Back to rgb: 3d tracking of hands and hand-object interactions based on short-baseline stereo. *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, 575-584.

Pisharady, P. K., Vadakkepat, P., y Poh, L. A. (2014). Hand posture and face recognition using fuzzy-rough approach. En *Computational intelligence in multi-feature visual pattern recognition: Hand posture and face recognition using biologically inspired approaches* (pp. 63–80). Singapore: Springer Singapore. Descargado de `https://doi.org/10.1007/978-981-287-056-8_5` doi: 10.1007/978-981-287-056-8_5

Simon, T., Joo, H., Matthews, I., y Sheikh, Y. (2017, July). Hand keypoint detection in single images using multiview bootstrapping. En *The ieee conference on computer vision and pattern recognition (cvpr).*

Sinha, A., Choi, C., y Ramani, K. (2016, June). Deephand: Robust hand pose estimation by completing a matrix imputed with deep features. En *The ieee conference on computer vision and pattern recognition (cvpr).*

Sridhar, S., Oulasvirta, A., y Theobalt, C. (2013, diciembre). Interactive markerless articulated hand motion tracking using rgb and depth data. En *Proceedings of the IEEE international conference on computer vision (ICCV).* Descargado de `http://handtracker.mpi-inf.mpg.de/projects/handtracker_iccv2013/`

Stenger, B., Thayananthan, A., Torr, P. H. S., y Cipolla, R. (2004). Hand pose estimation using hierarchical detection. En N. Sebe, M. Lew, y T. S. Huang (Eds.), *Computer vision in human-computer interaction* (pp. 105–116). Berlin, Heidelberg: Springer Berlin Heidelberg.

Tang, D., Yu, T., y Kim, T. (2013). Real-time articulated hand pose estimation using semi-supervised transductive regression forests. En *2013 ieee international conference on computer vision* (p. 3224-3231).

Tompson, J., Stein, M., Lecun, Y., y Perlin, K. (2014, septiembre). Real-time continuous pose recovery of human hands using convolutional networks. *ACM Trans. Graph.*, *33*(5). Descargado de `https://doi.org/10.1145/2629500` doi: 10.1145/2629500

Zhang, J., Jiao, J., Chen, M., Qu, L., Xu, X., y Yang, Q. (2016). 3d hand pose tracking and estimation using stereo matching. *CoRR*, *abs/1610.07214*. Descargado de `http://arxiv.org/abs/1610.07214`

Zimmermann, C., y Brox, T. (2017a). *Learning to estimate 3d hand pose from single rgb images* (Inf. Téc.). arXiv:1705.01389. Descargado de `https://lmb.informatik.uni-freiburg.de/projects/hand3d/` (https://arxiv.org/abs/1705.01389)

Zimmermann, C., y Brox, T. (2017b, Oct). Learning to estimate 3d hand pose from single rgb images. En *The ieee international conference on computer vision (iccv).*

Lista de Acrónimos y Abreviaturas

CNN Convolutional Neural Network.
RNN Recurrent Neural Network.